Grant Negotiation and Authorization Protocol (GNAP)
IETF109 Editors’ Update

Justin Richer
Aaron Parecki
Fabien Imbault
This is not an extension to OAuth 2
This is not “OAuth 3”
Moving Beyond OAuth 2.0

• OAuth 2 isn’t going anywhere
  – It’s been around for a decade
• But it has its limits
  – It’s been around for a decade
Design Considerations

- Protocol for negotiating access
- Methods for interacting with humans
- Validating and verifying the client software
- Methods for binding keys to message requests
- Data model of what’s being requested
Current State

• Design team initial draft accepted as starting point
  – Combination of XYZ and Xauth drafts
• Editors chosen and engaged
• Lots of notes about design decisions and potential choices going forward
  – Now being tracked in GitHub
• Some test implementations in the works
Design Team Output

- Single document
- Combined functional aspects of XYZ and Xauth drafts
- Cohesive protocol
- Not final output
  - Many decisions to be made
  - Lots of work ahead
Editor’s Notes

• Discussion points from design team meetings
  – Decisions, considerations, and options

• Currently inline in the document
  – Have been extracted to GitHub issues
  – Will be removed from inline text and replaced with links in next revision
Editors’ Process

- All changes through pull requests
  - Review required from at least one other editor for merge
- Link to issues and commits for decision tracking
- Substantive changes to be verified on the list
- Releases tagged in repository
Document Source

- Markdown source using kramdown-rfc2629
- Markdown checked in, built artifacts are not
  - Yes: .md
  - No: .xml, .html, .txt
- Repo includes Dockerfile and build script
Issue Discussion
Issue 29: Terminology

• What should we call things?
• How do we decide on names?
• How do we avoid endless months of bikeshedding?
RC/Client -> Client Instance

- Cryptographic identity
  - Distinct instance = access to a specific key
- Current spec text uses of “RC” and “client” are nearly always about a single client instance
Issues 44, 45, 46: Client Information

• Separate client instance from client software
  – OAuth 2 conflates these, we can detangle

• Can send either:
  – Instance identifier (like a “client_id”)
  – Client information block (like dynamic registration)
Collating Instances

• Provide mechanisms to tie together instances
  – Software statement, class identifier, others?

• Apply the same policies to a set of instances
  – Software version, platform
Use cases for client information

• Verifiable display information
• Provide means for runtime posture and attestation
  – What posture and attestations do we want?
  – What is needed at each request vs. first request?
• Third-party pre-registration services
Issue 67: Continuation

- Current response contains a URL an optional access token for continuing/managing the request
- Should access token be mandatory?
- Should access token be removed?
- Should URI be required to be unique?
"continue": {
  "access_token": {
    "value": "80UPRY5NM33OMUKMKSU",
    "key": true
  },
  "uri": "https://server.example.com/continue",
  "wait": 60
}
"continue": {
  "uri": "https://server.example.com/continue/80UPRY5NM33OMUKMKSU",
  "wait": 60
}
"continue": {
    "uri": "https://server.example.com/continue",
    "wait": 60
}
Benefits of Access Token Approach

• Authorization as an API
  – Uses same semantics as resource access
  – Client already knows how to do this
  – Similar to OAuth2 Dynamic Client Registration Management

• Allows many different deployment patterns for AS
  – Distributed AS by allowing unique and unrelated URIs
  – Stateless AS by packing request state into access token (or URI)
Benefits of Token-free Approach

- Fewer artifacts for a client to manage
- Different behaviors for talking to AS vs. RS
  - AS doesn’t have to manage special tokens
Issue 59: Interaction Bundles

• Current document:
  – Multiple interaction methods
  – One post-interaction method

• Alternative 1: couple post-interaction with single interaction method

• Alternative 2: allow multiple interaction bundles
"interact": {
    "redirect": true,
    "user_code": true,
    "callback": {
        "method": "redirect",
        "uri": "https://client.example.net/return/123455",
        "nonce": "LKLTI25DK82FX4T4QFZC"
    }
}
Alternative 1: Combined

"interact": {
  "redirect": {
    "callback": "redirect",
    "uri": "https://client.example.net/return/123455",
    "nonce": "LKLTI25DK82FX4T4QFZC"
  },
  "user_code": true
}

Alternative 2: Multiple explicit bundles

```
"interact": [
    {
        "redirect": true,
        "callback": {
            "method": "redirect",
            "uri": "https://client.example.net/return/123455",
            "nonce": "LKLTI25DK82FX4T4QFZC"
        }
    },
    {
        "redirect": 255,
        "user_code": true
    }
]
```

Single bundle passed as object (current syntax)
Alternative 2: Multiple explicit bundles

```
"interact": [
  {
    "Mode": "Method",
    "Method": {
      "redirect": true,
      "callback": {
        "method": "redirect",
        "uri": "https://client.example.net/return/123455",
        "nonce": "LKLTI25DK82FX4T4QFZC"
      }
    }
  },
  {
    "Mode": "Method",
    "Method": {
      "redirect": 255,
      "user_code": true
    }
  }
]
```
Issue 40: Token Request Syntax

• Current method allows only resource description
  – Simple for clients, analogous to “scope”
  – Awkward to send additional flags or fields

• Proposal: an “access_token” request structured object
Token Request Now

"resources": [
  {
    "type": "photo-api",
    "actions": [ "read", "write", "dolphin" ],
    "locations": [
      "https://server.example.net/",
      "https://resource.local/other"
    ],
    "datatypes": [ "metadata", "images" ]
  },
  "read",
  "bind_token",
  "multi_token"
]

"Token Flags"
Proposed syntax

"access_token": {
  "resources": [
    {
      "type": "photo-api",
      "actions": ["read", "write", "dolphin"],
      "locations": [
        "https://server.example.net/",
        "https://resource.local/other"
      ],
      "datatypes": ["metadata", "images"
      ],
      "read"
    },
    "key": true,
    "split": true,
    "label": "photo-token"
  ]
}
Proposed syntax (multiple)

"access_token": [
    {
        "key": false,
        "label": "reader",
        "resources": [ "read" ]
    },
    {
        "key": true,
        "label": "writer",
        "resources": [ "write" ]
    }
]
Issue 6: Polymorphism

- JSON has no inherent type restrictions
- Protocols (especially security protocols) benefit from predictable structures
- Polymorphism allows each field to have optimized expressions for different cases
  - Different options are naturally mutually-exclusive
Two equivalent requests

Object:
"resources": [
  {
    "type": "photo-api",
    "actions": [ "read" ],
    "datatypes": [ "metadata" ]
  },
  {
    "type": "photo-api",
    "actions": [ "write" ],
    "datatypes": [ "image-data" ]
  }
]

String:
"resources": [ "metadata", "update-image"

The AS decides how this is mapped
How can we support polymorphism?

• Use it only where it makes the most sense
• Who does it impact? Who can ignore it?
  – Optimize complexity away from client software
• Schema language for validation (informative only)
  – JSON Schema, CDDL, others?
  – Test suite
• Explicitly declare possible JSON types for all fields
What if we remove polymorphism?

• What would we do instead?
  – Mutual exclusivity of fields
  – Type fields for different objects
  – Container objects for single values

• What’s the complexity cost for the protocol?
  – Who pays the complexity?
Thank you!